Fast Mining of Distance-Based Outliers in High Dimensional Datasets*

Amol Ghoting, Srinivasan Parthasarathy, and Matthew Eric Otey
Department of Computer Science and Engineering
The Ohio State University, Columbus, OH 43210, USA.
Contact email: srini@cse.ohio-state.edu

November 9, 2005

Abstract

Defining outliers by their distance to neighboring data points has been shown to be an effective non-parametric approach to outlier detection. In recent years, many research efforts have looked at developing fast distance-based outlier detection algorithms. Several of these efforts report log-linear time performance as a function of the number of data points on many real life low dimensional datasets. However, these same algorithms are unable to obtain the same level of performance on high dimensional data sets since the scaling behavior is exponential in the number of dimensions. In this paper we present RBRP, a fast algorithm for mining distance-based outliers, particularly targeted at high dimensional data sets. RBRP is expected to scale log-linearly, as a function of the number of data points and scales linearly as a function of the number of dimensions. Our empirical evaluation verifies this expectancy and furthermore we demonstrate that our approach consistently outperforms the state-of-the-art, sometimes by an order of magnitude, on several real and synthetic datasets.

Keywords: Outlier detection, high dimensional datasets, approximate k-nearest neighbors, clustering.

1 Introduction

A common problem in data mining is that of automatically finding outliers or anomalies in a data set. Outliers are those points that are highly unlikely to occur given a model of the data. Since outliers and anomalies are rare, they can be indicative of bad data, faulty collection, or malicious content. Recently, researchers have applied outlier detection to tasks such as data cleaning [9], fraud detection [7], and intrusion detection [16].

There are several approaches to outlier detection. One approach is that of model-based outlier detection, where the data is assumed to follow a parametric (typically univariate) distribution [2]. Such approaches do not work well in even moderately high dimensional spaces and finding the right model is often a difficult task in its own right. To overcome these limitations, researchers have turned to various non-parametric approaches that use a point's distance to its nearest neighbor as a measure of unusualness [1, 13, 14].

Distance-based outlier detection has proven to be an effective non-parametric approach. However, the process is time consuming. The nested loop (NL) algorithm [13] typically requires $O(N^2)$ time, where N is the numbers of data points. This quadratic dependency in the number of data points (N) restricts the use of the NL algorithm on large data sets with millions of points.

To overcome this problem, in the past few years, researchers have proposed several algorithms to efficiently find distance-based outliers. Solutions range from the use of spatial index structures for fast *nearest neighbor* computation to partitioning the feature space with clustering [14]. Unfortunately, these approaches do not scale well with the number of dimensions [4, 13]. Consequently, for high dimensional data sets, solutions based on the simple NL algorithm are known to provide the best performance. Bay and Schwabacher [4] have shown that coupled with data randomization and a simple pruning rule, the NL algorithm provides the best known performance on large, high-dimensional data sets. While the worst case complexity of the NL algorithm continues to be $O(N^2)$, on several real and synthetic data sets, their approach achieves a complexity that is sub-quadratic (often well below quadratic but not quite log-linear), in the number of data points. Although their strategies do result in a significant reduction in computation, execution time does not scale linearly with the data set size, and the process can still be very time consuming.

In this paper, we further improve the scaling behavior of distance-based outlier detection on large high dimensional data sets. Specifically, we make the follow-

*This work is supported in part by NSF grants CAREER-IIS-0347662 and NGS-CNS-0406386.

ing contributions:

- We study the conditions under which the state-of-the-art distance-based outlier detection algorithm (due to Bay and Schwabacher [4]) is unable to provide near-linear time performance.
- We present a two phase algorithm for fast mining of distance-based outliers. In the first phase, the data set is pre-processed into bins such that points that are close to each other in space are likely to be placed in the same bin. This facilitates fast convergence to a point's *approximate nearest neighbors*. As we shall see, improving the performance of the NL algorithm only requires fast determination of a point's *approximate nearest neighbors*, and not its *nearest neighbors*. In the second phase, an extension of NL algorithm that operates over bins is used for fast determination of outliers.
- We demonstrate that our algorithm scales well to high dimensional data sets with millions of data points. Furthermore, we show that our algorithm outperforms the state-of-the-art distance-based outlier detection algorithm, sometimes by over an order of magnitude.

The rest of this paper is organized as follows. In Section 2, we present a background on state-of-the-art distance-based outlier detection. In Section 3, we present the shortcomings of the state-of-the-art, which is followed by RBRP, our two-phase outlier detection algorithm. We evaluate the performance of our algorithm in Section 4. Finally, we conclude in Section 5.

2 Distance-Based Outlier Detection

In distance-based outlier detection, one typically looks at the local neighborhood of a data point to find its nearest neighbors. If the nearest neighbors are relatively close, then the data point is considered to be normal, otherwise it is considered to be an outlier. The key benefit of defining outliers based on their neighborhood is that no parametric distribution needs to be defined to measure unusualness. The following are three popular definitions of distance-based outliers:

- Outliers are the data points for which there are fewer that p other data points within distance d [13].
- Outliers are the top n data points whose distance to the k^{th} nearest neighbor is greatest [14].
- Outliers are the top n data points whose average distance to the k nearest neighbors is greatest [1].

While there are several minor differences between these definitions, all these definitions use the nearest neighbor density estimate to determine if a point is an outlier. Researchers have developed a variety of approaches to find these outliers efficiently. Some approaches make use of KD-trees [5], R-trees [10], or X-trees [6] to find the nearest neighbors of each candidate point. These index structures are extremely efficient in finding the k nearest neighbors of a data point. Outlier detection algorithms that make use of these index structures can potentially scale as $O(NlogN)$, if one can find the k nearest neighbors of a data point in $O(logN)$ time. However, the time required to search through these index structures scales exponentially with the number of dimensions. Consequently, their usefulness is constrained to low dimensional spaces. Similarly, researchers have proposed partitioning the space into regions [13, 14], which allows for fast determination of nearest neighbors. These approaches are also affected by the curse of dimensionality and do not scale to high dimensional data [4].

The simple nested loop (NL) algorithm is known to give the best performance in high dimensional spaces [13]. The algorithm, in its simplest form, is presented in Table 1. The main idea in the NL algorithm is that for each data point in D, we keep track of its k closest neighbors as we scan the data set. When a data point's k^{th} closest neighbor has a distance that is less than the cutoff threshold, c, the data point is no longer an outlier, and we can proceed with the next data point. As we process more data points, the algorithm finds more extreme outliers, and the cutoff increases giving us improved pruning efficiency.

The state-of-the-art distance-based outlier detection algorithm, ORCA [4], uses the NL algorithm with a preprocessed data set. ORCA randomizes the data set (D) in linear time with constant amount of memory using a disk-based shuffling algorithm. This randomization allows the NL algorithm to process non-outlier points, which are the large majority, relatively quickly. The authors report sub-quadratic time performance (often well below quadratic but not quite log-linear) on several real and synthetic data sets.

3 Outlier Detection Algorithm

Before we present our outlier detection algorithm, we first attempt to characterize the shortcomings of ORCA.

3.1 Shortcomings of ORCA For expository simplicity, let us assume that we are interested in finding the top n outliers in a data set, where an outlier's score is equal to its distance from its nearest neighbor. Let

Procedure: Find Outliers
Input: k, the number of nearest neighbors; n, the number of outliers to be returned; D, the set of data points.
Output: O, the set of outliers.

```
begin
    c = 0 (c is the cutoff threshold)
    O = {}
    for each d in D
        Neighbors(d) = {}
        for each b in D such that b ≠ d
            if |Neighbors(d)| < k or Distance(b, d) < Maxdist(d, Neighbors(d))
                Neighbors(d) = Closest(d, Neighbors(d) ∪ b, k)
            endif
            if |Neighbors(d)| ≥ k and c > Distance(b, d))
                break
            end if
        end for
        O = TopOutliers(O ∪ b, n)
        c = MaxThreshold(O)
    end for
end
```

Note:
Maxdist(d, S) returns the maximum distance between d and an element in set S
Closest(d, S, k) returns the k nearest elements in S to d
TopOutlier(S, n) returns the top n outliers in S based on the distance to their kth nearest neighbor
MaxThreshold(S) returns the distance between the weakest outlier in S and its kth nearest neighbor

Table 1: The Simple Nested Loop Algorithm

us examine the number of distance computations that are required to process a data point (say x) that is not an outlier. One can think of this problem as a set of independent Bernoulli trials where one keeps drawing instances until one has a single success (one data point within the cutoff threshold). Let $\Pi(x)$ be the probability that a randomly selected data point lies within the cutoff threshold. Let Y be a random variable representing the number of trials required until we have a single success. The probability of obtaining a success on trial y, $P(Y = y)$, is given by:

$$P(Y = y) = \Pi(x) \times (1 - \Pi(x))^{(y-1)} \tag{3.1}$$

Therefore, the expected number of distance computation for the data point (x) that is not an outlier is given by:

$$E[Y] = \sum_{y=1}^{N} P(Y = y) \times y = \frac{1}{\Pi(x)} \tag{3.2}$$

In order to achieve near-linear time scaling behavior, the expected number of distance computation for a data point, $E[Y]$, must be a constant. While the cutoff threshold, c, is expected to increase as more data points are processed, on average $\Pi(x)$ must be constant. This is the central premise behind ORCA's near-linear time performance. However, as we shall see next, this does not always hold.

Again, for simplicity of explanation, let us assume that we have N uniformly distributed data points in an area of size $\sqrt{N} \times \sqrt{N}$. We seek to answer the following question: *If we pick a point x randomly in this area, what is the expected value of the cutoff threshold, c, such that $\Pi(x)$ will be constant?* Intuitively, for $\Pi(x)$ to be constant, the area of the circle with radius = c and center = x, πc^2, should scale as $O(N)$. In other words, c should scale as $O(\sqrt{N})$. Thus, when N data points are uniformly distributed in a $\sqrt{N} \times \sqrt{N}$, for near-linear time performance, ORCA requires the cutoff threshold to scale as $O(\sqrt{N})$. It is hard to expect the cutoff threshold to converge to such a large value quickly, not just for a uniformly distributed data set, but for any arbitrary data set. In summary, the limitations of ORCA are:

- It delivers near-linear scaling behavior only when the cutoff distance can quickly converge to a large value such that $\Pi(x)$ is a constant. This can occur only when the data set has a large number of outlying points, not just a few, such that the cutoff is raised high, relatively quickly. When the data sets consist of a mixture of a few distributions, with

not so many outlying points, ORCA's complexity is near quadratic [4].

- The number of comparisons needed per data point is still dependent on N, and is not a constant factor. Empirically, the authors show that the number of comparisons needed per data point shows sub-linear growth, and in some cases near-constant growth, in the number of data points, depending on the data set.

3.2 RBRP As pointed out in Section 2, in order to find distance-based outliers using the NL algorithm, one needs to find k data points that are within the cutoff threshold, c. We call these k data points *approximate nearest neighbors*. The key to fast outlier detection is to find the k approximate nearest neighbors of a data point in the shortest amount of time possible. This goal is different from most existing approaches that attempt to find the k nearest neighbors efficiently, which is more expensive. To tackle the shortcomings of ORCA and other existing approaches, we now present RBRP (Recursive Binning and Re-Projection), an algorithm for fast mining of distance-based outliers in high dimensional data sets. RBRP*, like* ORCA*, finds the top n outliers in the data set whose distance to the k^{th} nearest neighbor is the greatest.*

RBRP leverages the fact that one can determine the k approximate nearest neighbors of a data point in time that scales as $O(NlogN \times d)$, where d is the dimensionality of the data. RBRP is a two-phase algorithm. Figure 1 depicts the two phases of the algorithm. In the first phase, we recursively partition the data set into groups that are less than some user-defined *size*. We call these groups *bins*. Furthermore, within each bin, data points are reordered according to their projection along the principal component of the points in the bin. Phase 1 scales as $O(NlogN \times d)$ and allows us to efficiently find approximate nearest neighbors. In the second phase, we use an extension of the NL algorithm that finds outliers in a data set that is partitioned into bins. Phase 2 scales as $O(N \times d)$ in the average case, as we expect to find approximate nearest neighbors of a data point in constant time. Consequently, RBRP scales as $O(NlogN \times d)$. We will describe the two phases of RBRP next.

3.2.1 Phase 1 The goal of the first phase of RBRP is to partition the data set into bins such that points that are close to each other in space are likely to be assigned to the same bin. One natural candidate to generate such bins is to cluster the data using an algorithm such as K-means [11] to find a large number of small clusters. Each of the clusters can constitute a bin. However, this process requires us to specify the number of clusters, and does not guarantee equal-frequency binning, making it ineffective for our uses. Another possibility is to use a clustering algorithm such as BIRCH [17]. This is in some senses similar to the approach proposed by Ramaswamy *et al.* [14]. However, this approach will not scale to high dimensional data.

Our approach to partitioning the data set into bins is shown in Table 2. It is a recursive procedure known as divisive hierarchical clustering. At each stage in the recursion, we iteratively partition the data into k partitions. This iterative partitioning is akin to the partitioning step employed in the k-means [11] algorithm. Essentially, we start with k random centers, and assign each point to its closest center, creating k partitions. Next, we find k centers for these k partitions, and continue iteratively for a fixed number of iterations. Once we have finished with the iterations for each of these partitions, we proceed recursively if the size of the partition is greater than a user-defined threshold ($Binsize$). Such a binning strategy ensures that points that are close to each other in space are likely to be collocated in the same bin. In other words, approximate nearest neighbors are likely to be collocated in the same bin. As we are only concerned with finding the approximate nearest neighbors of a data point, and not its nearest neighbors, such a strategy works well in practice.

In Phase 2 of RBRP, we will sequentially scan through each bin to find the approximate nearest neighbors of a data point. To facilitate fast convergence to the approximate nearest neighbors during a sequential scan, we reorganize the data points in each bin as per their order in the projection along the principal component [12] of the bin. The principal component of a bin represents the axis of maximal variance. Such a reorganization within bins allows for fast convergence to approximate nearest neighbors when scanning sequentially through a bin. This is because we expect to find the approximate nearest neighbors of a data point in its neighborhood when data points are ordered as per their projection along their principal component. We also note that principal component analysis scales as $O(N \times d^3)$

Complexity analysis: Assuming a two-way partitioning at each step in the recursion, the recurrence relation for Phase 1 is given by:

$$T(N) = T(N-m) + T(m) + \theta(N) \tag{3.3}$$

In the worst case, if m is constant, then Phase 1 scales as $O(N^2)$. However, on average we expect m to have an $O(N)$ scaling behavior. Consequently, Phase 1 scales as $O(NlogN \times d)$ on average. Note, this argument is

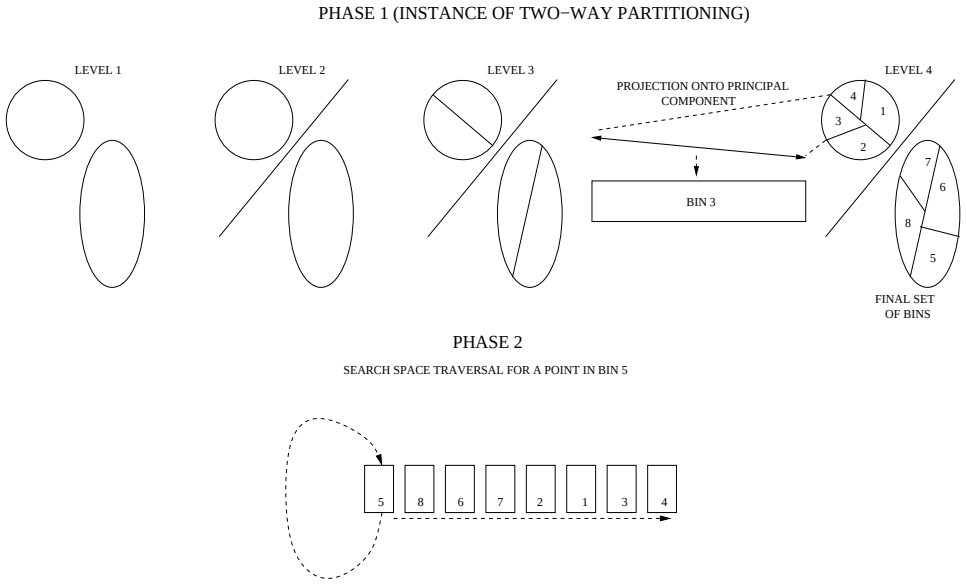

Figure 1: Phase1 and Phase2 of RBRP

akin to the average-case and worst-case complexity of the quick sort algorithm [8]. When employing a k-way partitioning at each step in the recursion, we expect similar scalability.

3.2.2 Phase 2 In Phase 2, we use an extension of the NL algorithm to find outliers in the data set that is organized into bins. For each data point, we start searching for approximate nearest neighbors beginning at the next consecutive location in the bin. Once the end of the bin has been reached, we wrap around to the start of the bin, and continue searching in the remainder of the bin. If the entire bin has been searched and k approximate nearest neighbors have not been discovered within this bin, we switch to the next closest bin, and continue searching for approximate nearest neighbors. This search continues iteratively until k approximate nearest neighbors are discovered.

Complexity analysis: The worst case time complexity of Phase 2 is $O(N^2)$. However, we expect to find the approximate nearest neighbors of a normal point in the very same bin. For outliers, we need to scan all of the bins, but this is expected to be a rare event, as number of desired outliers (n) is much smaller that the data set size (N). Therefore, we expect Phase 2 to scale as $O(N \times d)$. As Phase 1 scales as $O(NlogN \times d)$, we expect RBRP to scale close to $O(NlogN \times d)$.

At this juncture, we would like to point out that RBRP *will always discover the exact same set of outliers as* ORCA. The key difference between RBRP and ORCA is that when processing normal points, RBRP will discover the k approximate nearest neighbors in far less time than ORCA. For outliers, both ORCA and RBRP will need to scan the entire data set.

4 Experimental Results

4.1 Setup We evaluate our algorithm's performance on a Linux-based system with a 2.4 GHz Intel Pentium 4 processor and 1 GB of main memory. We report the wall clock time in order to capture both CPU and I/O time. All of the algorithms were implemented using C. While the source code for ORCA is not publicly available, the binary is publicly available. However, we chose not to use this binary for performance evaluation for the following reasons. First, it is not easy to compare performance between implementations that use different data set representations. For instance, we are not aware of the precision with which ORCA maintains floating point numbers. Second, we are not aware of other performance optimizations (such as improved memory and I/O managers) that may have been incorporated into the public implementation. The data set representation and the different performance optimizations can have a large impact on execution time. Therefore, we implemented our own version of ORCA, with the same set of optimizations and data set representation as RBRP's implementation. Our version scales like the public ORCA implementation and so comparisons provided henceforth will be based on our version.

We use several real and synthetic data sets for our analysis. These data sets are summarized in Table 4.

Procedure: Bin
Input: $Binsize$, the maximum size of a bin; k, the number of partitions; it, no. of iterations; D, data points to be binned.
Output: B, the set of bins.

```
begin
    c = {c_1, c_2, ..., c_k} (the set of k random centers)
    p = {p_1, p_2, ..., p_k} (the set of k partitions of D, initially empty)
    for it iterations
        Empty all k partitions in p
        for each d in D
            j = Closest(c, d)
            Insert(d, j)
        end for
        c = {}
        RecomputeCenters(c, p)
    end for
    for each p_i in p
        if size of p_i > Binsize
            Bin(Binsize, k, it, p_i)
        else
            Reorganize data points in p_i, ordered as per their projection along the principal component of p_i
            Add p_i to B
        end if
    end for
end
```

Note:
Closest(c, d) returns the index of the nearest elements in c to d
Insert(d, j) inserts point d in jth partition in p
RecomputeCenters(c, p) inserts k centers of partitions in p into c

Table 2: RBRP Phase 1

Procedure: Find Outliers
Input: k, the number of nearest neighbors; n, the number of outliers to be returned; D, the set of data points.
Output: O, the set of outliers.

```
begin
    c = 0 (c is the cutoff threshold)
    O = {}
    for each bin b in B
        for each d in b
            Neighbors(d) = {}
            for each t in B, ordered by increasing distance to b
                for each p in t such that p ≠ d
                    if |Neighbors(d)| < k or Distance(d, p) < Maxdist(d, Neighbors(d))
                        Neighbors(d) = Closest(d, Neighbors(d) ∪ p, k)
                    endif
                    if |Neighbors(d)| ≥ k and c > Distance(p, d))
                        break
                    end if
                end for
            end for
        end for
        O = TopOutliers(O ∪ b, n)
        c = MaxThreshold(O)
    end for
end
```

Note:
Maxdist(d, S) returns the maximum distance between d and an element in set S
Closest(d, S, k) returns the k nearest elements in S to d
TopOutlier(S, n) returns the top n outliers in S based on the distance to their kth nearest neighbor
MaxThreshold(S) returns the distance between the weakest outlier in S and its kth nearest neighbor

Table 3: RBRP Phase 2

Data set	Continuous Attributes	No. of Points
Corel Histogram	32	68,040
Covertype	55	5,81,012
KDDCup 1999	24	4,898,430
Mixed 30D	30	2,000,000
Uniform 30D	30	1,000,000
Ipums	128	2,000,000

Table 4: Data sets

They span a range of problems and have different types of features. We describe each in more detail.

- *Covertype* - This data set represents the type of forest coverings for 30×30 meter cells in the Rocky Mountain region. For each cell, the data contains the cover type, which is the dominant tree species, and additional attributes such as elevation, slope, and soil type.
- *Corel Histogram* - Each point in this data set encodes the color histogram of an image in a collection of photographs. The histogram has 32 bins corresponding to eight levels of hue and four levels of saturation.
- *Ipums* - This data set contains the responses from the 1990 Census in the United States.
- *KDDCup 1999* - This data set contains a set of records that represent connections to a military computer network where there have been multiple intrusions by unauthorized users. The raw TCP data from the network has been processed into features such as the connection duration, protocol type, number of failed logins, and so forth.
- *Mixed 30D* - This is a synthetic data set generated from a mixture of 30-dimensional normal and uniform distributions centered on the origin. The normal distribution is centered on the origin with a covariance matrix equal to the identity matrix. This data set contains one million data points from each of the distributions.
- *Uniform 30D* - This is a synthetic data set generated from a 30 dimensional uniform distribution centered on the origin and in the range [-1,1].

We obtained the Covertype, Corel Histogram, and KDDCup 1999 data sets from the UCI KDD Archive [3]. The Ipums data set was obtained from the IPUMS repository [15].

4.2 Scalability with increasing data set size Figures 2-4 show the total execution time to mine outliers on the six data sets as the number of data points are varied. Here, total execution time accounts for both the phases of RBRP. Each graph shows four lines. Two of these lines represent the expected execution time to mine the data set given a linear time algorithm and an $NlogN$ time algorithm. These lines are extrapolated from the first point in the line representing ORCA's execution time. The two remaining lines show the actual running times for RBRP and ORCA. The runs were set to mine the top 30 outliers with k set to 2.

RBRP outperforms ORCA on all the considered data sets. On the Covertype, Mixed 30D, and Uniform 30D data sets, RBRP outperforms ORCA by an order of magnitude. Furthermore, it shows improved scalability with increasing data set size when compared with ORCA. We can attribute these results to the fact that while RBRP incurs an $O(NlogN)$ pre-processing overhead, it can find outliers in near constant time per data point. For data sets that have a larger number of outlying data points, the cutoff threshold is able to increase quickly, and ORCA is able to give fairly good performance. This behavior can be seen on the Ipums data set. However, when the data set has a fewer number of outliers, the cut-off threshold does not grow fast. As a result, we get near quadratic performance for ORCA. This can be seen on the remaining data sets. The performance of RBRP is not affected as much by the slow decay in the cutoff threshold because of its improved search space, resulting in improved performance in all cases. Furthermore, Figures 2-4 indicate that RBRP does indeed scale as $O(NlogN)$. We note that on the Ipums and KDDCup 1999 data sets, it appears as though RBRP scales marginally better than $O(NlogN)$. This is simply due to the errors introduced during extrapolation.

4.3 Scalability with increasing number of nearest neighbors Figures 5-7 shows the total time to mine outliers on the six data sets as the number of nearest neighbors (k) are varied. For all these experiments, we mine outliers in the entire data set. Each graph shows actual running times for RBRP and ORCA. The runs were set to mine the top 30 outliers.

Both RBRP and ORCA exhibit linear scalability on all the considered data sets. Moreover, RBRP exhibits better scalability than ORCA with increasing k. This is attributed to the localized search for approximate nearest neighbors employed by RBRP. As k increases, for each normal point, we expect to see a constant increase in the number of bins that need to be searched. Unlike ORCA, RBRP is not affected by the slow decay in the cutoff threshold that occurs on most data sets. This is evident on all data sets except the Ipums data set. On the Ipums data set, the cutoff threshold converges to a large value relatively quickly.

Therefore ORCA and RBRP have comparable scaling performance on this data set.

5 Conclusion

In this paper, we presented RBRP, a two phase distance-based outlier detection algorithm targeted at high dimensional data sets. RBRP improves upon the scaling behavior of the state-of-the-art by employing an efficient pre-processing step that allows for fast determination of approximate nearest neighbors. RBRP is expected to scale as $O(NlogN \times d)$ on d-dimensional data sets with N data points. We validated its scaling behavior on several real and synthetic data sets. RBRP consistently outperforms *ORCA*, the state-of-the-art distance-based outlier detection algorithm, sometimes by an order of magnitude.

References

[1] F. Angiulli and C. Pizzuti. Fast outlier detection in high dimensional spaces. In *Proceedings of the International Conference on Principles of Data Mining and Knowledge Discovery*, 2002.

[2] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 1994.

[3] S. Bay. The UCI KDD archive. Irvine, CA: University of California, Department of Information and Computer Science, 1999.

[4] S. Bay and M. Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2003.

[5] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 1975.

[6] S. Berchtold, D. Keim, and H. Kreigel. The X-tree: an index structure for high dimensional data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1996.

[7] R. Bolton and D. Hand. Statistical fraud detection: A review. *Statistical Science*, 2002.

[8] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. Mac Graw Hill, 1990.

[9] D. Gamberger, N. Lavrac, and C. Groselj. Experiments with noise filtering in the medical domain. In *Proceedings of the International Conference on Machine Learning*, 1999.

[10] R. Guttmann. A dynamic index structure for spatial searching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1984.

[11] J. Hartigan. *Clustering Algorithms*. John Wiley and Sons, 1975.

[12] I. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[13] E. Knorr and R. Ng. Finding intensional knowledge of distance-based outliers. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1999.

[14] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large datasets. In *Proceedings of the International Conference on Management of Data*, 2000.

[15] S. Ruggles and M. Sobek. Integrated public use microdata series: Version 2.0, 1997.

[16] K. Sequeira and M. Zaki. ADMIT: Anomaly-based data mining for intrusions. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2002.

[17] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 1996.

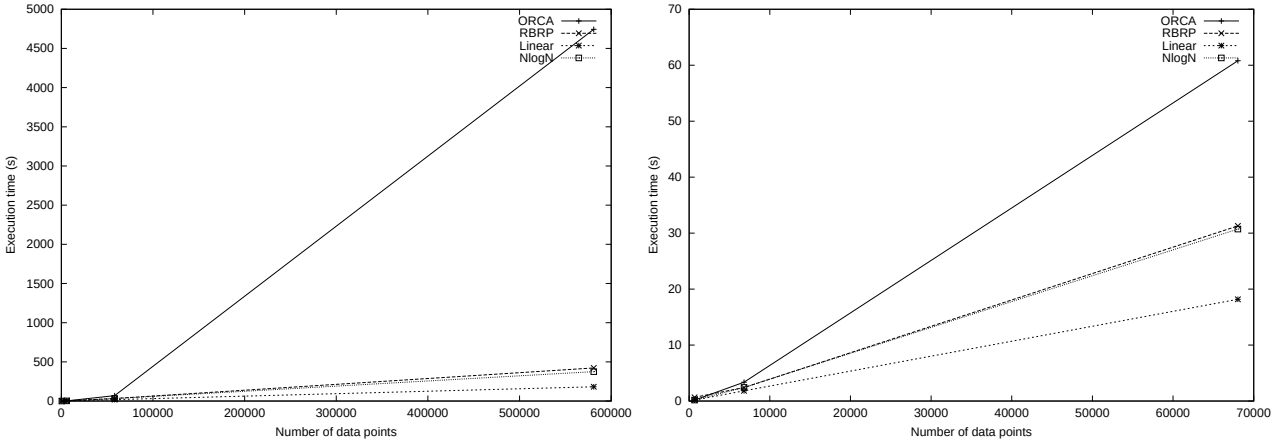

Figure 2: (a) Covertype (b) Corel histogram

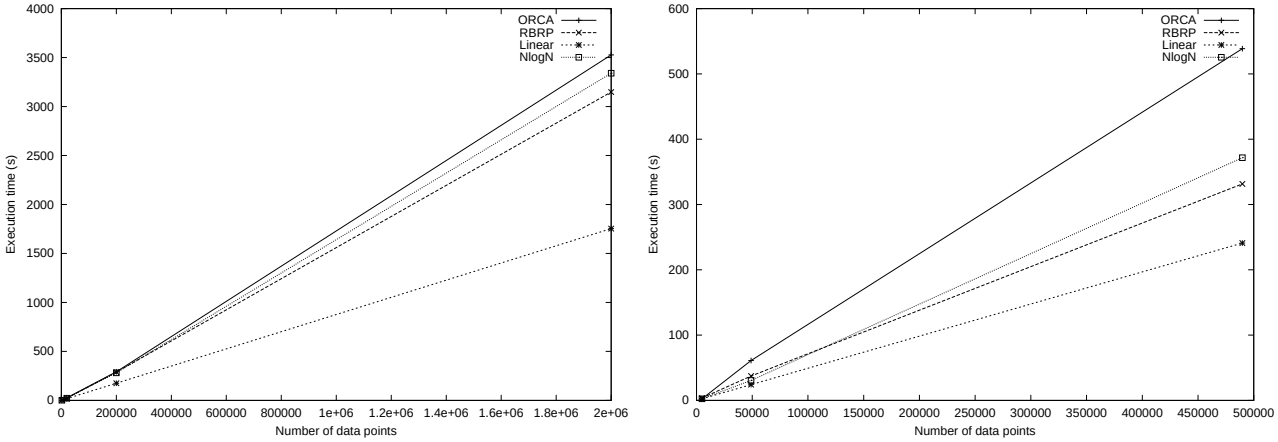

Figure 3: (a) Ipums (b) KDDCup 1999

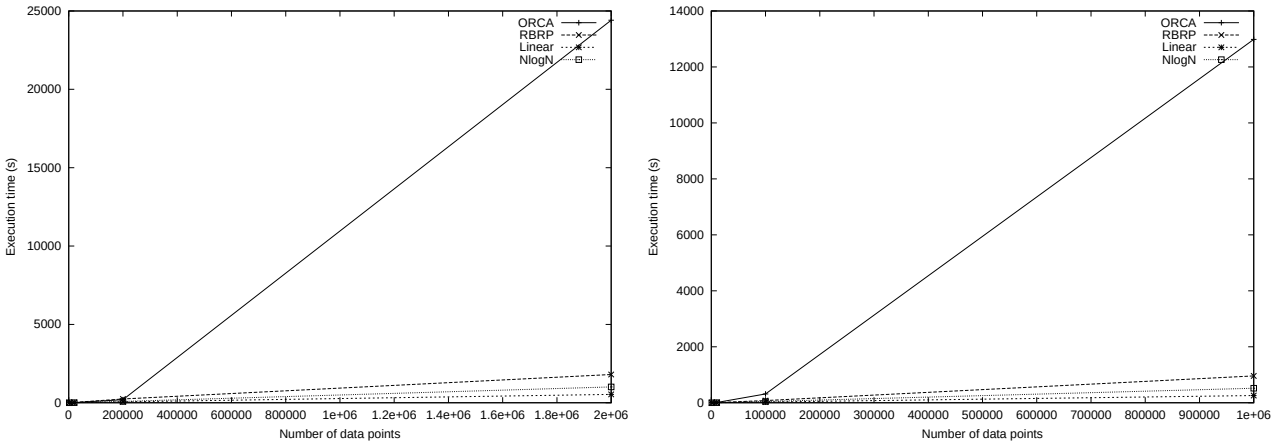

Figure 4: (a) Mixed 30D (b) Uniform 30D

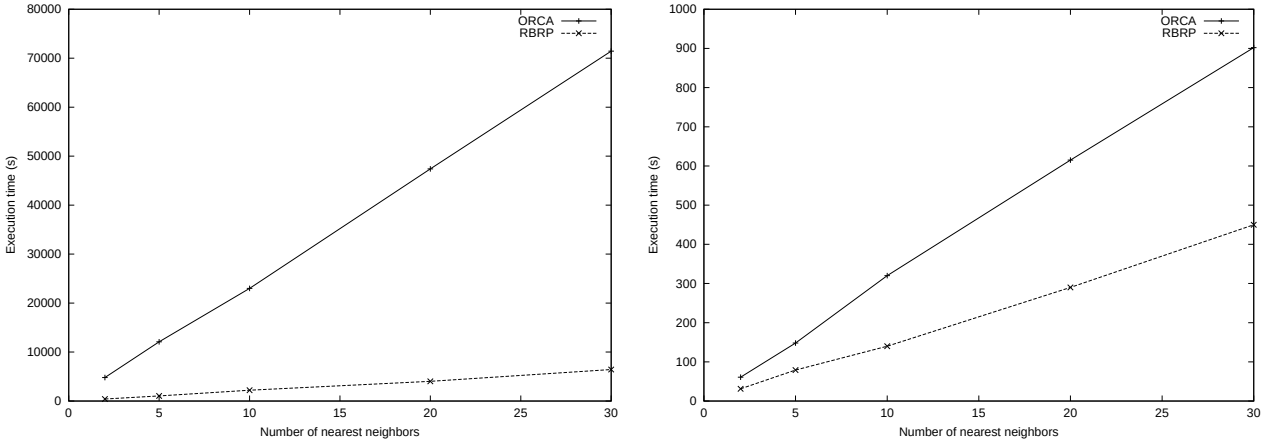

Figure 5: (a) Covertype (b) Corel histogram

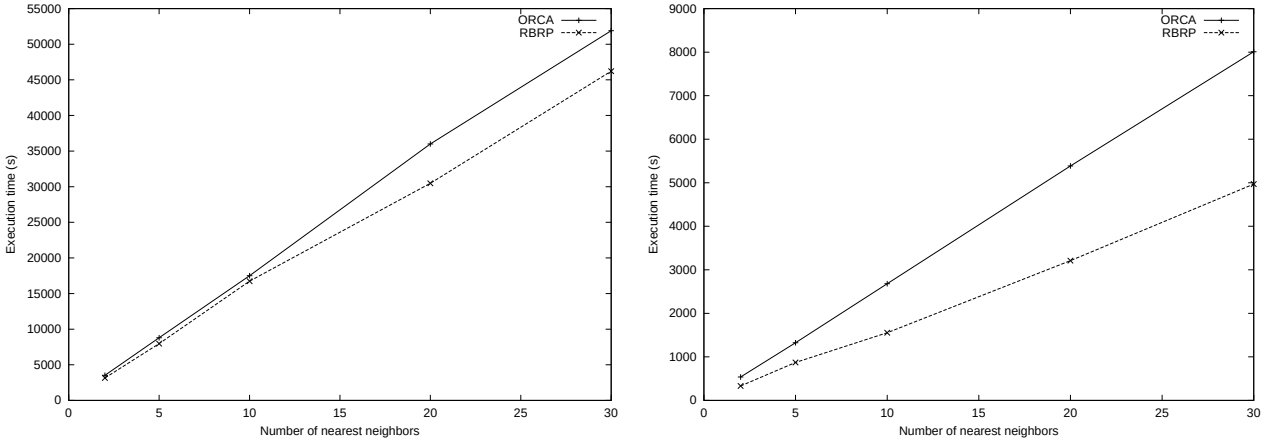

Figure 6: (a) Ipums (b) KDDCup 1999

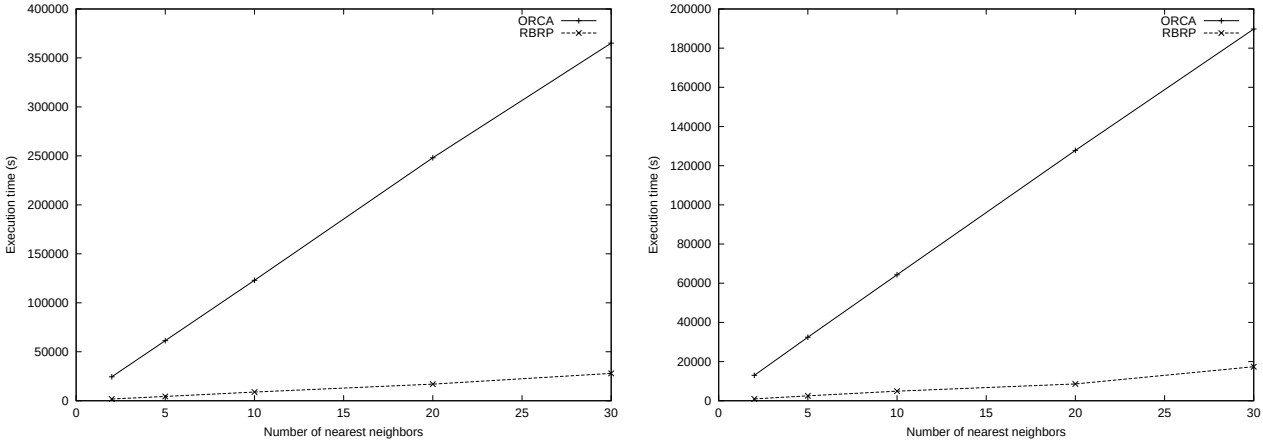

Figure 7: (a) Mixed 30D (b) Uniform 30D